18-100 Introduction to ECE
N21: Networking

Greg Kesden, Tom Zajdel, Liam Carden
Carnegie Mellon University

Connecting Computers Together

Networking is concerned with connecting computers together so that they can transfer data between each other. The I2C serial protocol discussed a few lectures ago is a hyper-local sort of networking, trading information between a microcontroller and a number of devices on a PCB. Modern computer networks span continents and in some cases have extended their reaches to satellites in outer space (and beyond).

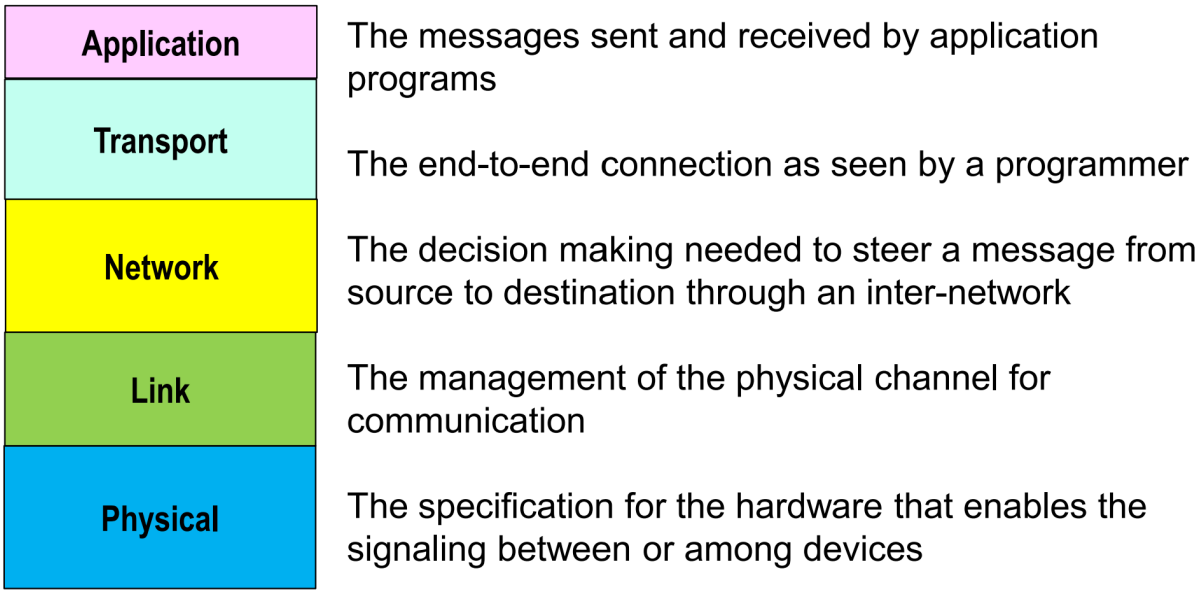

Figure 1: Overview of the Five-Layer Network Model

5-Layer Network Model

When discussing networks, we often like to break them down into five layers of functionality that address different issues in networking, although more layers are sometimes used.

1. **Physical Layer:** The physical means of transmitting signals between computers. Specifies physical structures like cable sizes, voltage levels, wire lengths, light wavelengths, signal frequencies, modulation techniques, and so on.
   Example protocols: WiFi 6 (802.11ax)

2. **Link Layer:** Manages the sharing of the physical layer resources within a **local area network (LAN)**. Provides the framing of data, identification of the sender and intended recipient, data collision management, error detection, and so on.
   Example protocols: Ethernet, Bluetooth.

3. **Network Layer:** Enables LANs to be assembled into an internetwork graph and **routes** data through this graph across networks. Determines the *path* for data, but does not send any data itself.
   Example protocols: IP (Internet Protocol).

4. **Transport Layer:** Establishes a virtual endpoint for data on the network: a Port. Also defines the quality of service of the connection between the two points on the network, specifying error-correction, acknowledgment between sender and receiver, etc.
   Example protocols: TCP, slower, more secure, and reliable; UDP, faster and less reliable.

5. **Application Layer:** Handles specifics based on application (data security/encryption, type of message), each associated with a Port established in the Transport Layer.
   Example protocols: HTTP, HTTPS, FTP, SSH.

1 Physical Layer

The physical layer is a hardware specification. It establishes the ability to signal from one station to another station. Once the physical layer has been established, it is possible to signal from one station to another. There must be a common structure to the messages so that both stations can understand each other.

Figure 2: The physical layer describes the properties of the *connection* between Station A and Station B. This link could be wireless (e.g. radio) or wired (e.g. ethernet). The **latency** of the connection is related to its physical length and any data processing required.

The wireless modulation methods we discussed in the wireless communication lectures (e.g. BPSK, 8-PSK, 64-QAM) are considered part of this specification. The physical requirements of the pull-up resistors and voltages used on an I2C bus are another example of physical layer concerns. In addition to these physical details, signal **latency** and **throughput** are of concern at this layer.

1.1 Latency

It takes time for a signal to propagate from A to B. The speed of this movement varies by the specific media, e.g. light via fiber optics or RF via air or electromagnetic energy via a wire. For the media that we use, the relevant speed is the "speed of light" through the media. If we send a message over a long distance, e.g. to China and back, this latency can be significant.

The amount of time it takes a message to go from $A \rightarrow B$ is called its one-way latency. The amount of time it takes to go from A to B and back, such as sending a message and having it acknowledged, is known as the **round trip latency** or **round trip time (RTT)**. Over short distances, most latency is caused by data processing time. Over long distances, the speed of light becomes more important.

1.2 Throughput/Data Rate

How much data we can send at a time is affected by the speed at which that data travels. Think about a modern highway:

- The more lanes there are the more cars can travel at a time.
- The shorter each car is, the more we can pack into the same space.
- The greater the speed limit, the faster the cars can go and the more quickly they can get there.

The speed limit affects **latency**; higher speeds means less latency. We measure latency, whether round-trip time or one-way time in seconds. The number of lanes and the size of the cars affect the **throughput**, i.e. the amount of work that can be done per unit of time. We measure throughput in terms of bits per second (bps).

We can achieve higher throughput by taking less time to clock each bit (each bit still takes the same amount of time to propagate along the wire), or by clocking more bits at the same time (e.g. using more wires/cables, a greater frequency range, or a denser encoding).

2 Link Layer

When using a network to transfer information, one must give structure to the data so that it may be understood. We call this structure **framing**. The link layer establishes that structure for a **local area network (LAN)**, a number of stations connected within relatively close proximity. In other words, the link layer structures and manages the use of the physical layer to enable useful communication.

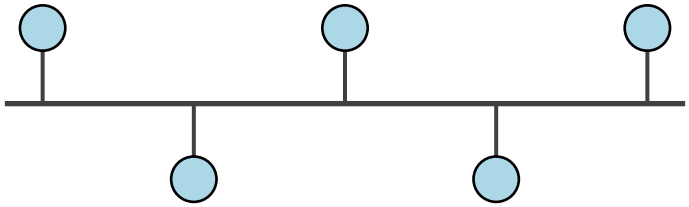

Figure 3: An example Local Area Network (LAN) graph. In this case, devices are connected to a common *bus*, like in I^2C or Ethernet. The Link layer's responsibility is to manage these shared physical connections, sending data frames from point to point along the LAN.

An I^2C network is an example of a local area network; a controller and a number of participants are physically connected over a 2-wire bus. Some management of the shared physical connections must be managed to avoid collisions, and this is part of the Link layer's specification. Data is sent as a *frame* of bits that includes a start condition, an address, data, and a stop condition. Without this standardized framing, the data is an unstructured stream of bits that cannot be interpreted.

As networks get larger, many users will need to share physical resources that connect the computers together. These resources represent data physically on the data link between computers, such as voltages on wires (e.g., I^2C) or waves in the radio spectrum (e.g. WiFi). There must be some management of these links to prevent data loss due to data collisions.

2.1 Framing

One of the most fundamental jobs of the link layer is to frame data. In other words, it breaks up the data that is to be sent into manageable units and provides structure to those units. Each frame normally has a beginning and an end. In addition to the data it carries, it normally carries **metadata** (information *about* the data) such as the identification of the sender and intended recipient, the size of the frame, etc.

Figure 4: Example dataframe on a local area network. There is a header and footer of metadata that *frame* the data.
Preamble - A sequence of alternating 1 and 0 to alert the computer to indicate the frame is next
DT - Data Type
FCS - Frame check sequence using cyclic redundancy code to check the fidelity of the data sent
Media access control (MAC) address - Hardware address

When this metadata is placed before the data, it is called a *header*. When it is placed after the data, is called a *footer*.

2.2 Station IDs

The link layer must identify all participants in a LAN. This is normally done with a station ID. Ethernet-derived protocols are very common and they call this the **Media Access Control ID (MAC ID or MAC address)**. This is just an identifier for a user of the shared physical channel, provided by the manufacturer. *No two devices on the same LAN may share the same MAC ID,* so devices are typically manufactured with unique MAC IDs.

In the I^2C protocol, all connected devices must each have a unique 7-bit address, which acts as a station ID.

2.3 Collision Management

Imagine that we have two stations at opposite ends of a wire. If one station sends to another, it will take some time for the data to get to the other side. If two messages cross, they damage each other in the **collision**. For this reason, when transmitting on a shared channel, stations generally listen first, to ensure that they don't hear another station transmitting. This is listening called **carrier sense**.

Because it can take time for signals to cross, merely sensing the carrier before transmitting isn't good enough. Stations must also detect collisions after they happen and recover from them. Minimizing such occurrences and recovering from them is the responsibility of the link layer.

Bit Distance

We can measure a network segment in three key ways:

1. **Physical length:** The physical distance of the link between stations (meters)
2. **Propagation time:** The time it takes a signal to traverse the physical length, i.e. the one-way latency (seconds)
3. **Bit distance:** The length of the data link *in bits*. Data rate in bits/second $\times$ a link's propagation time in seconds will describe how many bits will fit on the link at the same time (bits)

Bit distance vs Collision

For a station to detect collision it must still be transmitting at the time the collision occurs. This means that the minimum number of bits it can send, the *minimum frame size*, must equal the round trip bit distance. Why?

Imagine the collision occurs all the way at the other end of the wire, just at the destination. This is the worst-case scenario. It takes the one-way latency time for the collision to occur, then data from this now-corrupted frame needs the *same amount of time* to work its way back to the sender. In all, it took a full *round trip time* for the sender to "hear" the collision.

How does the sender "hear" the collision? It listens as it sends, and detects an incoming data frame from another station. From this, we see that the minimum frame size can be no smaller than the round-trip bit distance to enable collision detection.

Recovering From Collision

The typical way a sender recovers from a collision is to wait some amount of time and try again. But, it doesn't use a fixed timer. It waits a random amount of time. This is to avoid a situation where the two sides keep retrying – and recolliding – over and over and over again.[1] Common "Ethernet" link layers use **Carrier Sense Multiple Access with Collision Detection (CSMA/CD)** to manage collision. In other words, they listen both before transmitting (*carrier sense*) and while transmitting (*collision detection*).

[1] As a detail, "Ethernet" protocols use a random exponential backoff. In other words, a discussed, they wait a random amount of time before retrying after a collision. And, each time they re-collide, they exponentially increase the range from which they choose their retry time to try to spread out potentially colliding transmission over more and more time.

2.4 Local Area Networks (LANs)

A small homogeneous network is a **Local Area Network (LAN)**. Take a bunch of stations and connect them together to form a LAN. Maybe they are all connected to the same wire. Or, maybe they are all connected to the same network switch. Or, maybe they are all within radio signal range of each other. How do they talk? They broadcast. If there are only two stations, the station that receives the message is the intended recipient. However, in the more general case, every station hears the broadcast messages.

Each station has a station ID or LAN address. When a message is sent, it includes both the source and destination addresses. Although all stations might well hear all messages – they ignore those for which they are *not* the intended recipient.[2]

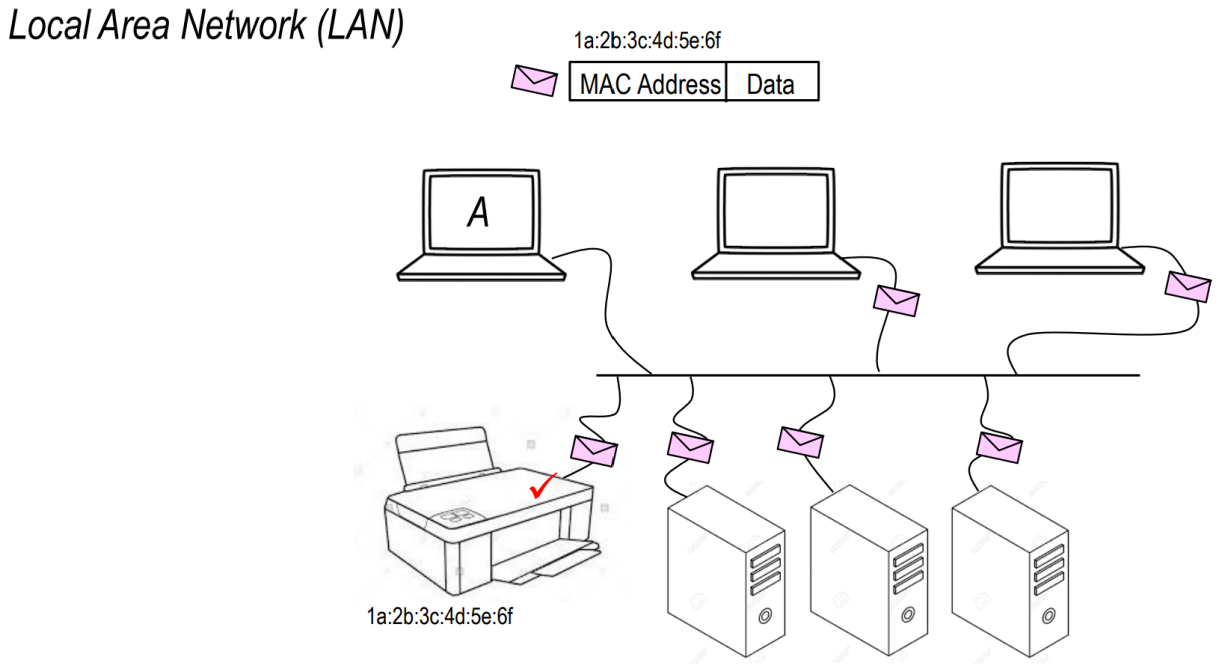

Figure 5: The MAC address is used to have the destination device accept the document and all other devices reject it.

[2]It is possible to snoop into other stations' messages; this is called "promiscuous mode." But, this is usually only done for diagnostic (or malicious) purposes.

The Size of a LAN is Self-Limiting

The size of a LAN is self-limiting, both in terms of physical size and also in terms of the number of stations.

1. *Physical size:* The longer a wire, the more attenuation – the signal is weakened as it travels farther and farther. The greater the distance through the air, the weaker the signal. In the end, there is only so far a phsyical distance that a signal can travel.
2. *Number of stations:* The more stations we have sharing a broadcast channel, the less network time exists per station. With modest use from a number of stations, the network could easily become clogged with collisions.

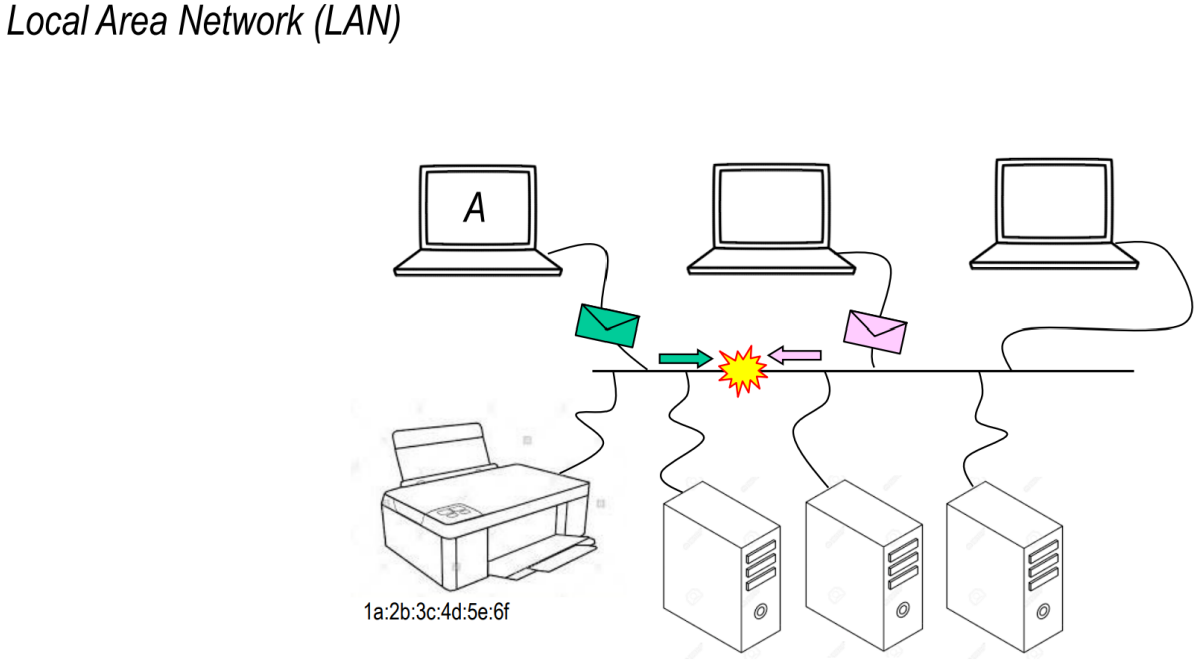

Figure 6: On this network, only one message is allowed at a time. Otherwise, a *collision* happens. During a message on the network, others have to wait.

Stretching LANs with Switches

It is possible to stretch the size of a LAN by using a **network switch**. The basic idea is that we can take a bunch of separate physical LANs and connect them together to form a larger logical LAN. The switches receive, temporarily store, and retransmit signals from one network to another, correcting the signal strength, noise, and timing, as they do. As switches transmit, they make note of the originating LAN. Then, if they later hear a message destined for that station, they send it only to that one LAN, not to all of the connected LANs.

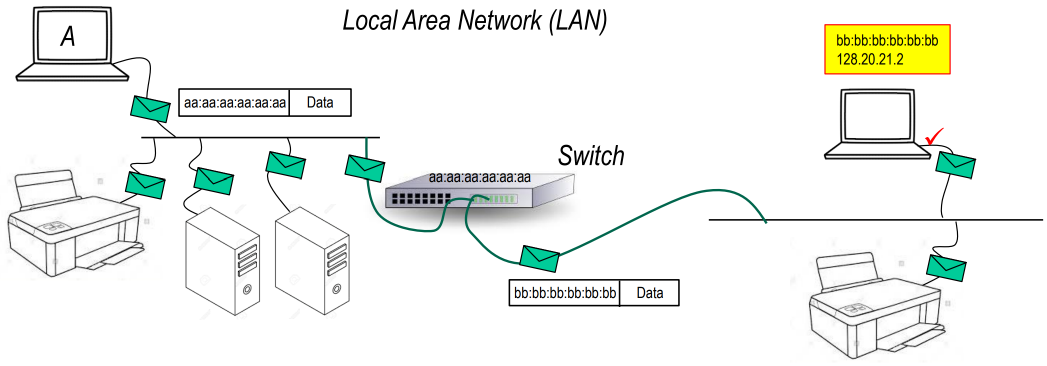

Figure 7: Switches can store, then forward messages from one network leg to another without flooding the whole network.

By dividing a larger LAN into multiple switch-controlled segments, the potential for collisions is reduced. A message can only collide on either the sender's segment or the receiver's segment. As long as the switch knows where the receiver is in the LAN, the other segments of the network are unaffected and can support additional transmissions.

Switches extend the size of LANs by a bit – but they aren't the *global* answer. Switches are largely limited by memory. There far too many stations on the planet for any single switch to remember them all. Even if memory were unlimited, it would be challenging to keep updating the constantly-evolving network structure of all devices connected together. The global *Internet* requires an inter-network, not a LAN.

3 Network Layer

Instead of scaling up LANs, we can recognize them as separate networks and efficiently communicate messages from one to the next, until we get from the *source network* to the *destination network*. The network layer manages the movement of messages across an **inter-network**, from LAN to LAN. It doesn't perform any communication. Communication is done via the physical layer as managed by the link layer. Instead, the network layer figures out the *route* a message should take. Messages are in the form of **packets** that contain routing information in their metadata.

As an analogy, you can think of the network layer as the "GPS Layer". The roads, which are managed for efficiency and safety, handle all of the actual movement of vehicles. But, there is a higher layer of management that figures out what path to take from start to finish and when to get on and off each road. In the networking world, this higher level of management is the network layer.

Congestion in a network can slow down different links at different times. And, just like roadways and bridges and construction, network links can become available and be lost over time. For this reason, the network layer makes routing decisions hop-by-hop, not all at once at the beginning. This is same the reason your car's GPS system can make changes along the way, correcting for traffic conditions or road closures.

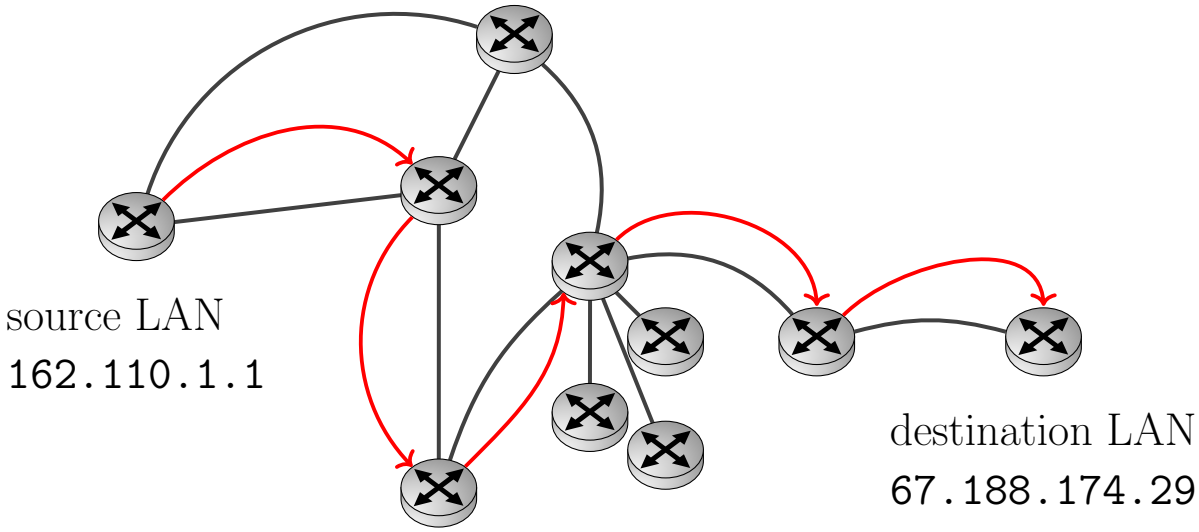

Figure 8: An example inter-network graph showing number of routers connected to each other. In this graph, each node represents a **router** handling traffic for a specific LAN. The Network layer's responsibility is to route packets of information from the source LAN to the destination LAN. In **packet-switched** network, these short packets of data are sent in small hops from network to network. The shortest path through this particular network is 4 hops long (see it?), but congestion on any particular network segment might result in a different route as pictured above (red arrows labeling the hops). The path the packet ultimately takes through the network is not known in advance.

Instead of viewing the entire Internet as one large, flat network, we are going to view it for what it is: a collection of individual networks. Step one is going to be routing packets from one network to another network. Once at the destination LAN, we'll worry about getting the packets to the right machine.

To achieve this, we are going to create a new **network address** - one that is structured so that it contains both a network number and the host number, rather than a flat address (i.e. station ID) that we've discussed so far. The station ID (i.e. MAC address) will still be used within a LAN - but we'll use this new **IP Address** to get from one network to another.

3.1 IP Address Details

The **Internet Protocol (IP)** is the most common network layer protocol. For our discussion, we are going to focus on IPv4, rather than its successor, IPv6. Both are in use today, but IPv4 addresses are simpler to understand.

IPv4 addresses are 32-bits wide, divided into four 8-bit numbers separated by decimal points. The first left-most bits represent the *network number*. They are the only bits required to route from network to network. The next, right-most, bits represent the host number and are only used once the packet makes its way to the destination network.

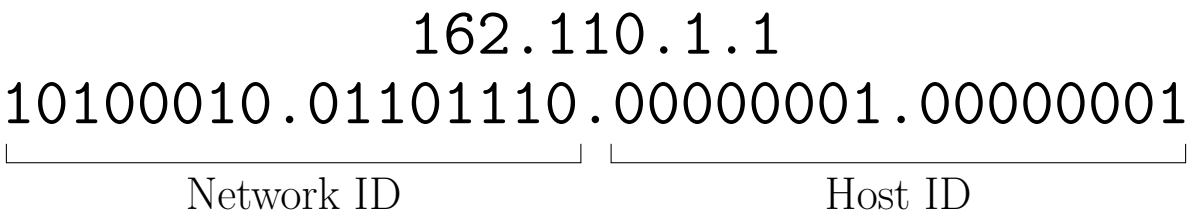

Figure 9: An example IPv4 address which consists of four bytes, first in dotted decimal notation (top), then in binary. This particular example is a Class B address, so the first 16 bits represent the network, and the last 16 bits represent the host.

In designing IP addresses, they could have decided that, in all cases, the left n-bits would be the network number. But, this would leave, in all cases, the right $(32 - n)$ bits to represent individual hosts. The problem is, of course, that not all networks are the same size. IBM's network is huge – not so much for Greg's Garage. There aren't enough addresses if we give enough IP addresses to Greg's Garage as we do a multi-national technology conglomerate.

So, what the designers of IP did was to create a few different classes of network addresses: small networks, medium networks, and large networks. You'll notice that the way they divide up the bits results in very few very large networks, a lot of mid-sized networks, and a huge number of small

networks:

- Class A (huge): 8 bits (network) + 24 bits (host), begins with `0`
- Class B (big): 16 bits (network) + 16 bits (host), begins with `10`
- Class C (small): 24 bits (network) + 8 bits (host), begins with `110`

Do you see a problem? Just considering class A networks, there are only 127 (the first bit must be `0`), each with $2^{24} = 16.7$ million hosts. In 2023, there are over *10 billion* internet-connected devices. IPv4 scheme has ran out of addresses...which is why IPv6 was developed, using 128-bit addresses enabling every device in existence to have its own address.

Adoption of IPv6 has been slow, because much of the internet runs on band-aid solutions developed in the 1990s that extended the life of IPv4[3]. Some day in the not-so-distant future, we're going to need all those IPv6 addresses.

3.2 How Internet Routing Works

At this point, we are viewing our inter-network as what it is - a collection of networks tied together. Tying these networks together are **routers**. Ultimately, when a message is sent from one host to another, one of two things is true:

- It is destined for a host on the same network.
- It is destined for a host on another network.

If it is destined for a host on the same network, there is no routing. The host, itself, looks at the destination IP address, notices that it is on the same network, and simply sends the message to the destination using the lower-level protocol. But, if it is destined for a different network, it sends it to the router instead.

[3]Classless Inter-Domain Routing (CIDR), which allows for more flexibility in the sizes of networks specified IP addresses, and Dynamic Host Configuration Protocol (DHCP) which allows IP addresses to be "recycled" if needed. The details are not critical in 18100, but they are quite interesting!

The router is a device that ties together several networks. It gets a message because the lower-level MAC address indicates it as the destination. But, it knows that it isn't the real destination, because the higher-level IP address indicates another recipient.

It masks off the host bits of the IP address, so it sees only the network number. Based on this, it looks in a table and forwards the message to one of the connected networks. If the destination lives on that connected network, it gets sent directly there using the lower-level protocol. Otherwise, the lower level protocol is still used - but to send it to another router, as described above.

3.3 Routing Protocols

It is important to note that these routers might be connected to many networks. It is even more important to note that these networks might form a graph, with multiple paths between destinations. And, yet more important to realize that there might be many, many hops from source to destination.

Given this, how do the routers know which way to send a packet so that it doesn't get lost or go around in circles? The answer is that the routers talk, and, based on that conversation, they build up two tables: one that describes the network, as a whole, known as the routing table, and one that describes exactly what the router should do, known as the forwarding table. We're going to leave the details of how these tables get built to 15-441. Especially since there are different protocols that get the job done and different strategies – and there are tons of interesting and subtle things about them.

4 Transport Layer

The transport layer establishes an end-to-end abstraction that is useful to the programmer. This abstraction becomes the **port**: a virtual endpoint for data sent across a network. This port number is tacked onto the IP address to identify the destination for a particular packet:

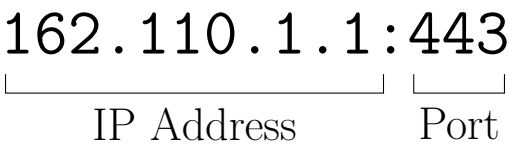

Figure 10: An IP address with intended port number appended after the colon. Port `443` is typically used by HHTPS, which sends encrypted data for web browsing.

Each port number is a different endpoint for a set of data on a user's system. For example, remote Minecraft gaming sessions use Port 25565 by default, while unsecured web traffic (HTTP) uses Port 80. The port hides the hop-by-hop nature of the network layer's routing process, providing a simple interface for programs to use with the network. Since multiple programs might be running on the same host, a port also enables program-to-program communication

In addition to these basic requirements, the Transport Layer must somehow answer the question, "What is a message, and how do we know when we have one?" For example, we often classify transport layers as being either:

- **Message-oriented:** Messages are sent like mail. When you get a message, it comes in a discrete chunk. Whatever is placed into the envelope when sent is exactly what is in the envelope when it is read. Envelopes, even those sent in series from the same sender to the same receiver are never merged.

- **Stream-oriented:** There really isn't the concept of a discrete message – there is the flow of data. Consider a phone conversation or radio broadcast. These don't come in envelopes. There can be periods of quiet, but there is no packaging or dividing line.

Protocols are often also classified in terms of their quality of service:

- **Best-effort:** Also like the post office, the protocol does its best, but makes no guarantees. Messages may be lost or delivered out of order.
- **Reliable:** The protocol will try diligently to resend anything that is not confirmed to be delivered.

As we'll talk about soon, unreliable protocols may, or may not, be **session-oriented**. A **session-oriented protocol** establishes a relationship between the sender and receiver before any data is exchanged. this session remains in place until it is closed. So, in some sense, the recipient knows to be waiting for communication. *Unreliable* protocols do not need to be session-oriented. But, *reliable* protocols need to be session-oriented so that the sender and receiver can coordinate what has, and what has not, been successfully received.

In the context of Internet protocols, the TCP/IP protocol suite, there are two general-purpose transport protocols:

- **User Datagram Protocol (UDP):** Unreliable, message-oriented.
- **Transport Control Protocol (TCP):** Reliable, stream-oriented.

UDP adds very little value of IP, itself – basically, it adds *port numbers*. It allows applications to be identified with ports so that messages, upon arriving at the destination, can be sent to the right program.

We'll talk about TCP next. It adds a lot of value. It adds streams and reliability. But, for today, I'd like to examine what it means – and does not mean – to be a **reliable protocol**.

4.1 Simple Reliability

Let's consider how we could create a **reliable protocol**. First, we add a sequence number to each message in the list of messages we intend to send. When we send a message, we wait for an acknowledgement (ACK) from the recipient. We know the maximum round-trip time and wait at least that long. If we don't get an ACK within that time, we assume the worst – the message got lost en route to the recipient. We resend and wait again for the ACK. There won't be any confusion, even if the same message is received twice because the sequence number will enable the duplicate to be detected and discarded.

To this end, it is important to note that only one message is in flight at a time. The time between the sending of a message and when its ACK is received is dead air. For this reason, this type of reliable protocol is often known as a stop-and-wait protocol.

4.2 Reliable vs. Unreliable

A reliable media certainly beats one that is not. But, as we now know, a reliable protocol is really just a diligent protocol. It tries, and tries, and tries some more.

But, this is not always desirable. In some cases, a late packet is worthless – and resending it just wastes network time. This is the case for many types of real-time communication, such as live video or audio, e.g. telephone calls or webcams.

What does one do with a 10-minute old syllable? If we delay the subsequent syllables by 10 minutes, the call is worthless. And, if we charge forward, we can't exactly introduce a stray word later. It is best to just let it go and hear a brief pause or pop. The same is true of video. We'd rather see a brief freeze and a jump in one part of the frame than have the whole thing delayed.

4.3 A Classic Parable: The Two Armies Problem

A classic story to illustrate the difference between reliable media and a reliable protocol (used on top of unreliable media) is the one of "Two Armies".

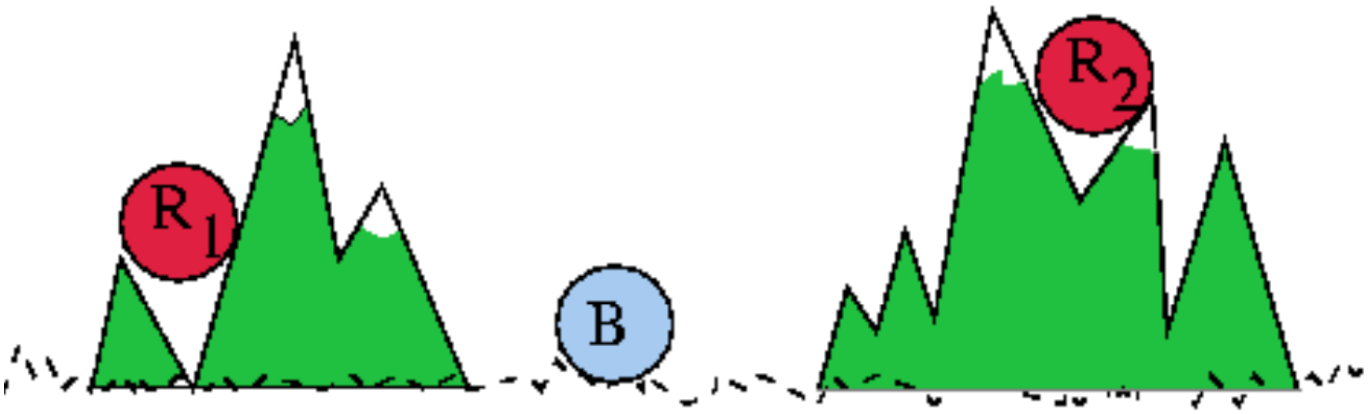

Figure 11: Red army can defeat blue army, but only if the two elements coordinate an attack at the same time, otherwise, it will be defeated

Imagine that there are two warring armies, the red army, and the blue army. The blue army is in a comfortable valley, where it is easily supplied. The red army is divided across two distant camps in the cold, barren, and hard-to-supply mountains. The blue army's best strategy is to wait for the red army to become hopeless and descend from the mountains hungry, cold, fatigued, divided, and easily defeated. If the encamped red armies attack one at a time, they'll be defeated, so the red army's best strategy is to coordinate across their camps and launch a synchronized attack, surrounding and defeating the blue army.

But, how can the red army coordinate? Imagine that one camp sends a messenger to the other camp, "Attack at dawn!" Problem solved, right? But what if the messenger while crossing the valley is killed by the blue army? Then the red army sending the messenger will attack alone and be defeated.

So, what if the message reads, "Attack at dawn! Please acknowledge!" Then, once the confirmation arrives back, the two camps can confidently attack, right? Well, what if the messenger gets killed on the way back? Then the sender doesn't know that the message has been received. And, even if the messenger makes its way through, the side sending the acknowledgment doesn't know the messenger made it through with the acknowledgment. They fear attacking alone and being defeated. Will they wait for the original camp to send another messenger to acknowledge this acknowledgement? When does it stop? There is always a final ACK.

This gets at what **reliable protocols** can do. They can keep sending a message until it gets ACKed or they get tired of retrying. They can often know if a message got through based on an ACK. They can't know if a message didn't get through based on missing an ACK. Although they can keep trying to send a message, it might never get through. And, when they keep retrying, they are introducing a delay. None of these costs are experienced with a "reliable media" which transmits messages correctly and in order.

4.4 Sliding Window Protocols

If, before moving on from sending one message to sending the next message, we wait for the send message to get to the other side, then wait for the other side to send an acknowledgment (ACK), then wait for that acknowledgment to come back, we are using what is called a **stop-and-wait protocol**. In other words, we are stopping and waiting for the other side to ACK a message before we move on to the next one. If we do not receive that ACK soon enough, we assume the message was lost and send the same message again (a message sequence number is used so that the other side knows it isn't a different, new message).

The good thing about this approach is that it enables us to have reliable transmission. We'll keep sending until we get the ACK (or time out

trying). The bad thing is that in the typical case when the message is not lost, we've left a lot of dead air time. Think about the time it takes to send the message. Now think about the following ratio:

$$\frac{\texttt{Message\_time}}{\texttt{Round\_trip\_time}}$$

This is the fraction of the available network time that we can use. At any given bit rate, we can also think of this in terms of the message length in bits and the round-trip bit distance of the network.

There is something that we can do about this: use a **sliding window protocol**. We can prepare a buffer that has the same bit length as the round-trip bit distance and break it into segments, sending one segment at a time in sequence. If all goes well, just as we send the last sequence and run out of things to send, we receive the ACK for the 1st thing in the buffer. This can be removed from the buffer, making room for something new to be placed in the buffer and sent – we don't need to stop sending. Once we reach steady state, we get one ACK back, which slides the buffer and makes room for something new, for each message that is sent. We can use all of the network time.

If a message or ACK is ever lost, we'll have to pause, resend the message, and wait for the ACK – doing that until it the ACK is received (or we give up). This condition is called the head of line blocking because everything else in the buffer might be ACKed (checked off as received by the recipient), but if the 1$^{\text{st}}$ thing in the buffer is not ACKed, the buffer can't be slid, and we can't put a new thing in.

Meanwhile, on the receiver's side, arriving segments are put into the buffer and the buffer can be "slid", giving data to the program that is using it, every time there is a sequence of ACKed segments at the front of the buffer. Should segments arrive out of order, the buffer serves to put them into order before handing them to the program.

5 Application Layer

The application layer is just the software layer that is the ultimate user of the network. If two different components of the system need to talk to each other, they need to have some set of messages that they know how to send and understand. The specifics of the messages vary by application. For example, there are different needs for...

- sending video and sound snippets from video conferencing (SIP)
- requesting Web objects from a Web server via a browser (HTTP/S)
- managing a mailbox and sending and receiving mail (SMTP)
- updating the universe and players in a game (no standard protocol)

For example, a request for a Web page made via the HTTP application protocol might look like this:

```
GET http://www.ece.cmu.edu/ HTTP/1.0
Host:  www.ece.cmu.edu
```

And a request to send an email message via the SMTP protocol might look like this:

```
MAIL FROM: <gkesden@andrew.cmu.edu>
RCPT TO: <gkesden@gmail.com>
DATA
From:  ‘‘Greg Kesden’’ <gkesden@andrew.cmu.edu>
To:  ‘‘Greg Kesden’’ <gkesden@gmail.com>
Subject:  SMTP Example

Greetings aaand welcome, everybody!
```

Quick Review

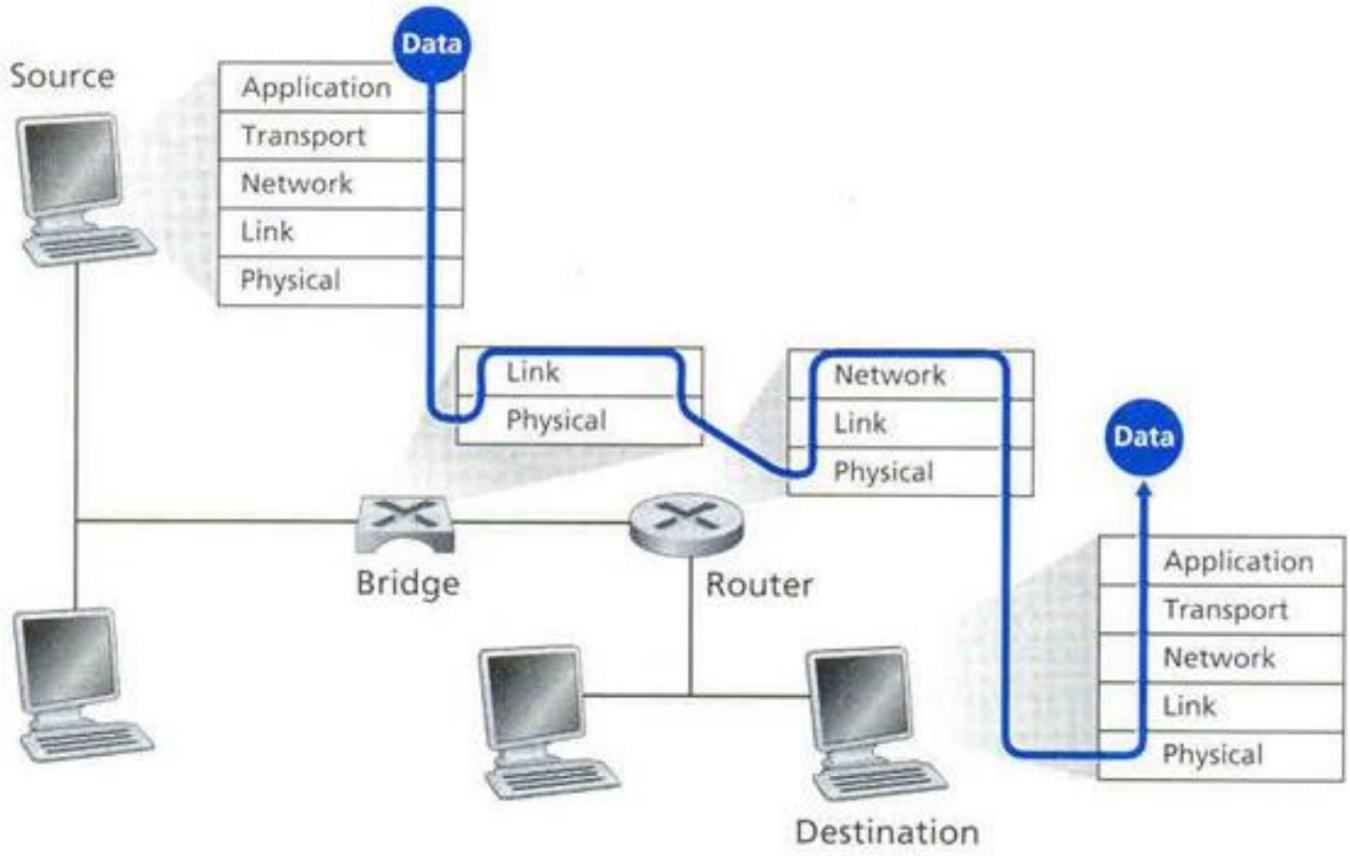

Figure 12: Summary of the layered structure

- **Application Layer:** The details of the messages and structures used by a particular application.
- **Transport Layer:** Establishment of endpoints and other services commonly used by programmers.
- **Network Layer:** Decisions about which way to move packets from network to network across an inter-network.
- **Link Layer:** Management of stations sharing the same channel.
- **Physical layer:** Voltages, connector shapes, power levels, light colors, etc.